Unidad 2.

# 1.- Elementos del lenguaje PHP.

Algunos de los elementos que se usan en el lenguaje, como las variables y tipos de datos, comentarios, operadores y expresiones.

Los delimitadores "<?php y ?>" o "<?= y ?>".

Los programas escritos en PHP, además de encontrarse estructurados normalmente en varias páginas suelen incluir en una misma página varios bloques de código.

Por ejemplo:

|  |
| --- |
| <body>  <?php $a=1; ?>  <p>Página de prueba</p>  <?php $b=$a; ?>  … |

La variable $a mantiene el valor.

Recomendación:

* Para los archivos que van a ser solamente de código PHP, es decir NO vamos a incluir marcas HTML es recomendable no cerrar el guion php.
* El IDE Apache Netbeans nos facilita la tarea de edición de código HTML y PHP con el uso de plantillas de código.

## 1.1.- Generación de código HTML.

Existen varias formas de generar contenido en la página web a partir del resultado de la ejecución de código PHP. La forma más sencilla es usando echo, que no devuelve nada (void), y genera como salida el resultado de evaluar el parámetro o parámetros de entrada y convertir el resultado a cadena de caracteres.

Una forma equivalente a usar **echo** para generar código de la página dinámica sería usar la etiqueta **<?=** y **?>**. A continuación, se muestra dos líneas de código que generan el mismo resultado.

|  |
| --- |
| <?= "La edad es $edad" ?>  <?php echo "La edad es $edad"; ?> |

Otra posibilidad es **print**. La diferencia principal entre **print** y **echo**, es que **print** sólo puede recibir un parámetro y devuelve siempre 1.

|  |
| --- |
| int print (string $arg); |

Tanto **print** como **echo** no son realmente funciones, por lo que no es obligatorio que pongas paréntesis cuando las utilices.

**prinft** (print con formato) es otra opción para generar una salida desde PHP. Puede recibir varios parámetros, el primero de los cuales es siempre una cadena de texto que indica el formato que se ha de aplicar. Esa cadena debe contener un **especificador** de **conversión** por cada uno de los demás parámetros que se le pasen a la función, y en el **mismo orden** en que figura en la lista de parámetros.

|  |
| --- |
| <?php  $ciclo="DAW";  $modulo="DWES";  print "<p>";  printf("%s es un módulo de %d curso de %s", $modulo, 2, $ciclo);  print "</p>";  ?> |

Cada especificador de conversión va precedido del caracter **%** y se compone de las siguientes partes:

* **signo** (opcional). Indica si se pone signo a los números negativos (por defecto) o también a los positivos (se indica con un signo +).
* **relleno** (opcional). Indica que carácter se usará para ajustar el tamaño de una cadena. Las opciones son el carácter 0 o el carácter espacio (por defecto se usa el espacio).
* **alineación** (opcional). Indica que tipo de alineación se usará para generar la salida: justificación derecha (por defecto) o izquierda (se indica con el carácter -).
* **ancho** (opcional). Indica el mínimo número de caracteres de salida para un parámetro dado.
* **precisión** (opcional). Indica el número de dígitos decimales que se mostrarán para un número real. Se escribe como un dígito precedido por un punto.
* **tipo** (obligatorio). Indica cómo se debe tratar el valor del parámetro correspondiente. En la siguiente tabla puedes ver una lista con todos los especificadores de tipo.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Imagen que contiene Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Existe una función similar a **printf** pero en vez de generar una salida con la cadena obtenida, permite guardarla en una variable: **sprintf**.

|  |
| --- |
| $txt\_pi = sprintf("El número PI vale %+.2f", 3.1416); |

## 1.2.- Funciones de control de valores de una variable.

Existen tres funciones booleanas que devuelven un valor lógico (true o false) dependiendo del valor y estado de la variable que se envía como parámetro de entrada.

Las funciones son las siguientes:

* **is\_null (mixed $var): bool** : Comprueba si la variable dada es nula. No asignada, Nula, no existe.
* **isset(mixed $var, mixed $... = ?): bool** : Determina si una variable está definida y no es **NULL**.
* **empty(mixed $var): bool** : determina si una variable está vacía. Una variable se considera vacía si no existe o su valor tiene una conversión implícita al valor booleano **false**.
  + "" (una cadena vacía)
  + 0 (0 como un integer)
  + 0.0 (0 como un float)
  + "0" (0 como un string)
  + NULL
  + FALSE
  + array() (un array vacío)
  + $var; (una variable declarada, pero sin un valor)

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamenteTabla

Descripción generada automáticamente

## 1.3.- Manejo de fechas. Uso de DateTime

Desde la versión 5.2.2, PHP ofrece mecanismos modernos y potentes para crear fechas y convertir entre formatos, usando la clase **DateTime** de PHP.

Para crear objetos de la clase DateTime utilizaremos el constructor de la clase. **new DateTime().**

Para establecer nuestro propio formato deberemos de emplear el método estático de la clase DateTime, este método nos vendrá muy bien cuando trabajemos con bases de datos como MySQL.

|  |
| --- |
| $dateTime = DateTime::createFromFormat('Y-m-d', "2020-12-31");  var\_dump($dateTime); |

Formatos de DateTime: <https://www.php.net/manual/en/datetime.formats.php>

Zonas horarias con DateTimeZone: <https://www.php.net/manual/en/timezones.php>

Zona horaria por defecto: **date\_default\_timezone\_set**('Europe/Madrid');

En el fichero de configuración **php.ini** en el parámetro **date.timezone** se puede establecer. <https://www.php.net/manual/es/class.datetimezone.php>

Una vez creado el objeto DateTime que representa un fecha y hora en una zona horaria concreta podemos expresarlo de muchas maneras formateándolo de la manera que nos interese.

|  |
| --- |
| public **DateTime::format**(string $format): string |

**Caracteres para componer la cadena de formato**

**Carácter Resultado**

d día del mes con dos dígitos.

j día del mes con uno o dos dígitos ( sin ceros iniciales ).

z día del año, comenzando por el cero ( 0 = 1 de enero ).

N día de la semana ( 1 = lunes, ..., 7 = domingo ).

w día de la semana ( 0 = domingo, ..., 6 = sábado ).

l texto del día de la semana, en inglés (Monday, ..., Sunday).

D texto del día de la semana, solo tres letras, en inglés ( Mon, ..., Sun ).

W número de la semana del año.

m número del mes con dos dígitos.

n número del mes con uno o dos dígitos ( sin ceros iniciales ).

t número de días que tiene el mes.

F texto del día del mes, en inglés (January, ..., December).

M texto del día del mes, solo tres letras, en inglés ( Jan, ..., Dec ).

Y número del año.

y dos últimos dígitos del número del año.

L 1 si el año es bisiesto, 0 si no lo es.

h formato de 12 horas, siempre con dos dígitos.

H formato de 24 horas, siempre con dos dígitos.

g formato de 12 horas, con uno o dos dígitos ( sin ceros iniciales ).

G formato de 24 horas, con uno o dos dígitos ( sin ceros iniciales ).

i minutos, siempre con dos dígitos.

s segundos, siempre con dos dígitos.

u microsegundos.

a am o pm, en minúsculas.

A AM o PM, en mayúsculas.

r fecha entera con formato RFC2822.

Los objetos DateTime pueden **compararse** con los operadores de comparación habituales como si fueran valores numéricos ( >< == ).

Se puede calcular la **diferencia** entre dos objetos **DateTime** utilizando el siguiente método.

|  |
| --- |
| public **DateTime::diff**(DateTimeInterface $datetime2, bool $absolute = false): DateInterval |

La diferencia entre las fechas se devuelve como una instancia de la clase **DateInterval** que representa un intervalo temporal.

|  |
| --- |
| $hoy = new DateTime('today');  $ayer = new DateTime('yesterday');  $interval = $hoy->diff($ayer);  echo $interval->format('Hace %d día') |

Propiedades de la clase DateInterval: <https://www.php.net/manual/es/class.dateinterval.php>

Por último, es posible realizar cierta aritmética con las fechas representadas por los objetos DateTime. En concreto se puede sumar, restar fechas. Estas operaciones se pueden realizar con los métodos:

|  |
| --- |
| public **DateTime::add**(DateInterval $interval): DateTime Añade el intervalo de tiempo a la fecha del objeto DateTime.  public **DateTime::sub**(DateInterval $interval): DateTime Sustrae el intervalo de tiempo a la fecha del objeto DateTime. |

|  |
| --- |
| $hoy = new DateTime('today'); echo $hoy->format('Y-m-d'), "</br>"; // 2022-11-01  $hoy->add(new DateInterval('P2D')); echo $hoy->format('Y-m-d'), "</br>"; // 2022-11-03  $hoy->sub(new DateInterval('P2D')); echo $hoy->format('Y-m-d'), "</br>"; // 2022-11-010 |

También podemos **modificar** el valor de fecha y hora almacenado en un objeto DateTime. De estas operaciones podemos deducir que los objetos DateTime **son mutables**. Si quisiéramos trabajar con objetos inmutables tendríamos que usar objetos de la clase **DateTimeImmutable**. Las operaciones realizadas sobre objetos inmutables devuelven nuevos objetos y nunca modifican los objetos originales.

El método que permite modificar un objeto DateTime es:

|  |
| --- |
| public DateTime::modify(string $modify): DateTime |

Por ejemplo:

|  |
| --- |
| $hoy = new DateTime('today');  echo $hoy->format('Y-m-d'), "<br>"; // 2022-11-01  $hoy->modify('-2 days');  echo $hoy->format('Y-m-d'), "<br>"; // 2022-10-30 |

Asociado con el API orientado a objetos ofrecido por PHP está la clase DatePeriod que representa un periodo de fechas y permite la iteración sobre un conjunto de fechas y horas que se repite a intervalos regulares.

|  |
| --- |
| $comienzo = new DateTime('2022-08-01');  $fin = new DateTime('2022-08-31');  $intervalo = new DateInterval('P2D');  $periodo = new DatePeriod($comienzo, $intervalo, $fin);  foreach ($periodo as $fecha) { // Devuelve fechas desde la fecha inicial cada do  echo $fecha->format("Ymd") . "<br>";  } |

Clase DatePeriod: <https://www.php.net/manual/es/class.dateperiod.php>

**DEPRECADO**

Si vas a trabajar con fechas y deseas que el formato y los nombres de días y meses aparezcan en castellano debes establecer la configuración regional a los valores españoles y utilizar la función **strftime**: <https://www.php.net/manual/es/function.strftime.php>

|  |
| --- |
| setlocale(LC\_ALL, 'es-ES.UTF-8');  date\_default\_timezone\_set('Europe/Madrid');  $ahora = new DateTime();  $fecha = strftime("Hoy es %A, %d de %B de %Y y son las %H:%M:%S", $ahora );  echo $fecha; |

Las marcas de tiempo o Unix *timestamps*, se tratade un entero que representa un punto temporal expresado en número de segundos transcurridos desde el 1 de enero de 1970.

|  |
| --- |
| $timestamp = strtotime('Mon, 12 Dec 2011 21:17:52 +0000');  $dt = new DateTime('@' . $timestamp); |

También es posible extraer y establecer el valor del objeto DateTime usando losmétodos:

* public setTimestamp(int $timestamp): DateTime Establece la fecha y la hora en base al Unix *timestamp.*
* public getTimestamp(): int Obtiene el valor de Unix *timestamp* asociado al objeto.

Además, es posible usar un abanico de funciones relacionadas con los *timestamps*:

* mktime( int $hour = date("H"), int $minute = date("i"), int $second =date("s"), int $month = date("n"), int $day = date("j"), int $year =date("Y"), int $is\_dst = -1): int Obtiene la marca de tiempo de una fecha
* strtotime(string $datetime, ?int $baseTimestamp = null): int|false Convierte una descripción de fecha/hora textual en inglés a un Unix *timestamp*.
* time(): int Devuelve el Unix *timestamp* actual.
* date(string $format, ?int $timestamp = null): string Da formato al Unix*timestamp* actual.

Existe un API procedural compuesto por repertorio extenso de funciones de fecha que puede consultarse en el siguiente enlace: <https://www.php.net/manual/es/ref.datetime.php>

## 1.4.- Variables especiales de PHP.

PHP incluye unas cuantas variables internas predefinidas que pueden usarse desde cualquier ámbito, por lo que reciben el nombre de **variables super globales**. Ni siquiera es necesario que uses **global** para acceder a ellas.

1. **$\_SERVER**. Contiene información sobre el entorno del servidor web y de ejecución.

|  |  |
| --- | --- |
| **Valor** | **Contenido** |
| **$\_SERVER['PHP\_SELF']** | guion que se está ejecutando actualmente. |
| **$\_SERVER['SERVER\_ADDR']** | dirección IP del servidor web. |
| **$\_SERVER['SERVER\_NAME']** | nombre del servidor web. |
| **$\_SERVER['DOCUMENT\_ROOT']** | directorio raíz bajo el que se ejecuta el guión actual. |
| **$\_SERVER['REMOTE\_ADDR']** | dirección IP desde la que el usuario está viendo la página. |
| **$\_SERVER['REQUEST\_METHOD']** | método utilizado para acceder a la página (**'GET'**,**'HEAD'**, **'POST'** o **'PUT'**) |

1. **$\_GET**, **$\_POST** y **$\_COOKIE** contienen las variables que se han pasado al guión actual utilizando respectivamente los métodos **GET** (parámetros en la url), HTML **POST** y Cookies HTML.
2. **$\_REQUEST** junta en uno solo el contenido de los tres arrays anteriores, **$\_GET**, **$\_POST** y **$\_COOKIE**.
3. **$\_ENV** contiene las variables que se puedan haber pasado a PHP desde el entorno en que se ejecuta.
4. **$\_FILES** contiene los ficheros que se puedan haber subido al servidor utilizando el método **POST**.
5. **$\_SESSION** contiene las variables de sesión disponibles para el guion actual.
6. **$GLOBALS** Es un array asociativo que contiene las referencias a todas las variables que están definidas en el ámbito global del script. Los nombres de las variables son las claves del array.

Variables superglobales: <https://www.php.net/manual/es/language.variables.superglobals.php>

## 1.5.- Tipos de datos compuestos.

En PHP puedes utilizar dos tipos de datos compuestos: el **array** y el **objeto**.

Un **array** es un tipo de datos que nos permite almacenar varios valores. Cada miembro del array se almacena en una posición a la que se hace referencia utilizando un valor clave o índice.

* Los **índices** pueden ser numéricos o simbólicos (cadenas de caracteres).
* Los **valores** de un array no tienen por qué ser del mismo tipo.
* Los arrays con índices **numéricos** se denominan arrays numéricos y los arrays con índices simbólicos se denominan **asociativos**.

Puedo transformar cualquier valor simple en un array mediante un *casting* forzado (array) y obtengo un array numérico de un único elemento en el índice 0 correspondiente al valor transformado.

La función **print\_r** es similiar a **var\_dump** pero en este caso muestra el valor de una array.

A partir de PHP 5.4 también se puede usar la sintaxis de array corta, la cual reemplaza **array()** con [ ].

Para hacer referencia a los elementos almacenados en un array, tienes que utilizar el valor clave entre corchetes: $mod1DAW [3], $mod1DAWAcronimo [PR]

Los arrays anteriores son vectores, esto es, arrays unidimensionales. En PHP puedes crear también arrays de varias dimensiones almacenando otro array en cada uno de los elementos de un array.

|  |
| --- |
| $DAW = [  "1DAW" => ["PR" => "Programación", "BD" => "Bases de datos", "LM" => "Lenguaje de Marcas", "ED" => "Entorno de Desarrollo", "SSII" => "Sistemas Informáticos", "FOL" => "Formación y Orientación Laboral"],  "2DAW" => ["DAW" => "Despliegue de Aplicaciones Web", "DWES" => "Desarrollo Web en Entorno Servidor", "DWEC" => "Desarrollo Web en Entorno Cliente", "DIW" => "Diseño de Interfaces Web", "EIE" => "Empresa e Iniciativa Emprendedora", "ING" => "Inglés"]  ];  print\_r($DAW); |

Para hacer referencia a los elementos almacenados en un **array multidimensional**, debes indicar las claves para cada una de las dimensiones: **$DAW ["2DAW"] ["DWES"]**

No es necesario que indiques el tamaño del array antes de crearlo. Ni siquiera es necesario indicar que una variable concreta es de tipo array. Simplemente puedes comenzar a asignarle valores:

|  |
| --- |
| $mod1DAW [0] = "Programación";  $mod1DAWAcronimo ["PR"] = "Programación"; |

Ni siquiera es necesario que especifiques el valor de la clave. Si la omites, el array se irá llenando a partir de la última clave numérica existente, o de la posición **0** si no existe ninguna:

|  |
| --- |
| $mod1DAW [] = "Programación"; |

## 1.5.1.- Recorrer arrays.

Las **cadenas de texto** o **strings** se pueden tratar como arrays en los que se almacena una letra en cada posición, siendo **0** el índice correspondiente a la primera letra, **1** el de la segunda, etc.

Para recorrer los elementos de un array, puedes usar un bucle específico: **foreach()**.

Utiliza una variable temporal para asignarle en cada iteración el valor de cada uno de los elementos del array.

Puedes usarlo de dos formas:

* Recorriendo sólo los elementos.
* Recorriendo los elementos y sus valores clave de forma simultánea.

|  |
| --- |
| foreach ($mod1DAWAcronimo as $clave => $modulo) {  echo "$clave: $modulo", "</br>";  } |

## 1.5.2.- Funciones de arrays.

Consulta: <https://www.php.net/manual/es/ref.array.php>

Las funciones más interesantes en el manejo de arrays:

* **array\_key\_exists**(mixed $key, array $array): bool Verifica si el índice o clave dada existe en el array.
* **array\_keys**(array $array): array Devuelve todas las claves de un array o un subconjunto de claves de un array.
* **array\_product**(array $array): number Calcula el producto de los valores de un array.
* **array\_sum**(array $array): number Devuelve la suma de los valores de un array.
* **array\_slice**( array $array, int $offset, int $length = null, bool $preserve\_keys = false): array Extrae una parte de un array.
* **array\_splice**( array &$input, int $offset, int $length = 0, mixed $replacement = array()): array Elimina una porción del array y la reemplaza con otra cosa.
* **array\_shift**(array &$array): mixed Quita un elemento del principio del array.
* **array\_unique**(array $array, int $sort\_flags = SORT\_STRING): array Elimina los valores duplicados de un array.
* **array\_values**(array $array): array Devuelve todos los valores de una array con índices numéricos consecutivos con base 0.
* **array\_fill**(int $start\_index, int $num, mixed $value): array Llena un array con valores.
* **count**(mixed $array\_or\_countable, int $mode = COUNT\_NORMAL): int Cuenta todos los elementos de una array.
* **in\_array**(mixed $needle, array $haystack, bool $strict = false): bool Comprueba si un valor existe en un array.
* **array\_search**(mixed $needle, array $haystack, bool $strict = false): int|string|false Busca un valor determinado en un array y devuelve la primera clave correspondiente en caso de éxito.
* **array\_rand**(array $array, int $num = 1): mixed Seleccionar una o más claves aleatorias de un array.
* **implode**(string $separator, array $array): string Une elementos de una array en un string.
* **explode**(string $delimiter, string $string, int $limit = PHP\_INT\_MAX): array Divide un string en un array de strings.

Para comprobar si una variable es de tipo array, utiliza la función **is\_array()**.

## 1.5.2.- Ordenación de arrays

Una de las acciones más interesantes a realizar sobre los arrays es la ordenación de sus elementos y/o sus claves. La librería de funciones de array ofrece un buen número defunciones de ordenación para conseguir distintos resultados. Todas las funciones que se presentan a continuación modifican el array cambiando sus elementos según el orden solicitado. A continuación, se presentan las funciones más útiles:

* **sort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena los valores de un array en base a las opciones que se pasan como parámetro.
  + SORT\_REGULAR - compara elementos normalmente (no cambia los tipos)
  + SORT\_NUMERIC - compara elementos de forma numérica
  + SORT\_STRING - compara elementos como cadenas
  + SORT\_LOCALE\_STRING - compara elementos como cadenas, basándose en la configuración regional en uso. Utiliza la configuración regional, la cual puede cambiarse usando setlocale().
  + SORT\_NATURAL - compara elementos como cadenas usando el "orden natural".
  + SORT\_FLAG\_CASE - se puede combinar (OR a nivel de bits) con SORT\_STRING o SORT\_NATURAL para ordenar cadenas de forma insensible a mayúsculas/minúsculas.
* **rsort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena un array en orden inverso.
* **asort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena un array y mantiene la asociación de índices.
* **arsort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena un array en orden inverso y mantiene la asociación de índices.
* **ksort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena una array por clave.
* **krsort**(array &$array, int $sort\_flags = SORT\_REGULAR): bool Ordena un array por clave en orden inverso.

Tabla

Descripción generada automáticamente

# 2.- Estructuras de control.

Los **guiones PHP** se construyen en base a sentencias. Utilizando llaves, puedes agrupar las sentencias en conjuntos, que se comportan como si fueran una única sentencia.

Para definir el flujo de un programa en PHP, al igual que en la mayoría de los lenguajes de programación, hay sentencias para dos tipos de estructuras de control:

* **sentencias** **condicionales**, que permiten definir las condiciones bajo las que debe ejecutarse una sentencia o un bloque de sentencias; y
* **sentencias** de **bucle**, con las que puedes definir si una sentencia o conjunto de sentencias se repite o no, y bajo qué condiciones.

## 2.1.- Condicionales.

Las sentencias condicionales son sentencias que alteran el flujo de ejecución haciendo que se ejecute un bloque u otro según la evaluación de una determinada condición.

* **if / elseif / else.** 
  + La sentencia if permite definir una expresión para ejecutar o no la sentencia o conjunto de sentencias.
  + Si la expresión se evalúa a true, la sentencia se ejecuta. Si se evalúa a false no se ejecutará.
  + Cuando el resultado de la expresión sea false, puedes utilizar else para indicar una sentencia o grupo de sentencias a ejecutar en ese caso. Otra alternativa a else es utilizar elseif y escribir una nueva expresión que comenzará un nuevo condicional.
  + Cuando las sentencias actúen sobre una única sentencia, no será necesario usar llaves. Se utilizará llaves cuando el condicional actúe sobre más de una sentencia.

|  |
| --- |
| if ($motor == 1) {  $descripcion = "El motor es de Gasolina";  } elseif ($motor == 2) {  $descripcion = "El motor es Diesel";  } else {  $descripcion = "Error, el tipo de motor NO es válido";  } |

* **Switch**
  + Es similar a enlazar varias sentencias if comparando una misma variable con diferentes valores. Cada valor va en una sentencia **case**. Cuando se encuentra una coincidencia, comienzan a ejecutarse las sentencias siguientes hasta que acaba el bloque switch, o hasta que se encuentra una sentencia **break**. Si no existe coincidencia con el valor de ningún case, se ejecutan las sentencias del bloque **default**, en caso de que exista.

|  |
| --- |
| $motor = 3;  switch ($motor) {  case 1:  $descripcion = "El motor es de Gasolina";  break;  default:  $descripcion = "Error, el tipo de motor NO es válido";  } |

## 2.2.- Bucles.

Los bucles o sentencias repetitivas son estructuras que permiten repetir una secuencia de sentencias mientras se de cierta condición.

* **while**: Usando while puedes definir un bucle que se ejecuta mientras se cumpla una expresión. La expresión se evalúa antes de comenzar cada ejecución del bucle.

|  |
| --- |
| $a = 1;  while ($a < 8){  $a += 3;  }  echo $a; // el valor obtenido es 10 |

* **do / while**: Es un bucle similar al anterior, pero la expresión se evalúa al final, con lo cual se asegura que la sentencia o conjunto de sentencias del bucle se ejecutan al menos una vez.

|  |
| --- |
| $a = 5;  do{  $a -= 3;  }while ($a > 10);  print $a; // el bucle se ejecuta una sola vez, con lo que el valor obtenido es 2 |

* **for:** Son los bucles más complejos de PHP. Al igual que los del **lenguaje C**, se componen de tres expresiones:

|  |
| --- |
| for (expr1; expr2; expr3) {  sentencia o conjunto de sentencias;  } |

* + La primera expresión, **expr1**, se ejecuta solo una vez al comienzo del bucle.
  + La segunda expresión, **expr2**, se evalúa para saber si se debe ejecutar o no la sentencia o conjunto de sentencias. Si el resultado el **false**, el bucle termina.
  + Si el resultado es **true**, se ejecutan las sentencias y al finalizar se ejecuta la tercera expresión, **expr3**, y se vuelve a evaluar **expr2** para decidir si se vuelve a ejecutar o no el bucle.

Puedes anidar cualquiera de los bucles anteriores en **varios** **niveles**. También puedes usar las sentencias **break**, para salir del bucle, y **continue**, para omitir la ejecución de las sentencias restantes y volver a la comprobación de la expresión respectivamente. **break** acepta un argumento numérico opcional que indica de cuántas estructuras anidadas circundantes se debe salir y de igual manera **continue** también acepta un argumento numérico opcional, que indica a cuántos niveles de bucles encerrados se ha de saltar al

final.

# 3.- Funciones.

Las **funciones** tienen una utilidad similar: **nos permiten asociar una etiqueta** (el nombre de la función) **con un bloque de código** a ejecutar. Además, al usar funciones estamos ayudando a estructurar mejor el código. Como ya sabes, las funciones permiten crear variables locales que no serán visibles fuera del cuerpo de las mismas.

Muchas funciones están incluidas en el núcleo y se pueden usar directamente. Otras muchas se encuentran disponibles en forma de extensiones, y se pueden incorporar a lenguaje cuando se necesitan.

Con la distribución de PHP se incluyen varias extensiones. Para poder usar las funciones de una extensión, tienes que asegurarte de activarla mediante el uso de una directiva **extensión** en el fichero **php.ini**. Muchas otras extensiones no se incluyen y antes de poder utilizarlas tienes que descargarlas.

## 3.1.- Definición de funciones

Para llamar a una función, basta con poner su nombre y unos paréntesis.

Para crear tus propias funciones, deberás usar la palabra **function**.

|  |
| --- |
| function nombre\_de\_la\_funcion($parametro1, $parametro2, ...) {  // Cuerpo de la función  // Puedes realizar operaciones y retornar un valor si es necesario  return $resultado;  } |

* **function**: Esta es la palabra clave que indica que estás definiendo una función.
* **nombre\_de\_la\_funcion**: Debes reemplazar esto con el nombre que deseas darle a tu función. El nombre debe seguir las **reglas** de nomenclatura de PHP y no debe contener espacios ni caracteres especiales, excepto el guion bajo (\_).
* **($parametro1, $parametro2, ...):** Los parámetros son **variables** que puedes pasar a la función. Estos son opcionales y se colocan entre paréntesis. Puedes definir tantos parámetros como necesites, separándolos por comas. Los parámetros permiten que la función acepte valores externos con los que pueda trabajar.
* **{}:** El **cuerpo** de la función se encuentra dentro de un par de llaves. Aquí es donde escribes el código que se ejecutará cuando la función sea llamada.
* **$resultado:** Este es un ejemplo de una variable que puedes usar para almacenar el resultado de la función. Puedes usar **return** para devolver un valor específico como resultado de la función.

En PHP puedes utilizar una función **antes** íncluso de ser definida en el código, excepto cuando la función está definida en una rama de una setencia condicional, aunque esta práctica es poco habitual.

## 3.2- Argumentos.

Los argumentos se indican en la definición de la función como una **lista** de **variables** separada por comas. No es necesario indicar el tipo de cada argumento, al igual que no es necesario indicar si la función va a devolver un valor y el tipo de este (si una función no tiene una sentencia **return**, devuelve **null** al finalizar su procesamiento).

Al definir la función, puedes indicar **valores por defecto** para los argumentos, de forma que cuando hagas una llamada a la función puedes no indicar el valor de un argumento; en este caso se toma el valor por defecto indicado.

Puede haber valores por **defecto** definidos para varios argumentos, pero en la lista de argumentos de la función todos ellos deben estar a la derecha de cualquier otro argumento sin valor por defecto.

En los ejemplos anteriores los argumentos se pasaban **por valor**. Esto es, cualquier cambio que se haga dentro de la función a los valores de los argumentos no se reflejará fuera de la función. Si quieres que esto ocurra debes definir el parámetro para que su valor se pase **por referencia**, añadiendo el símbolo **&** antes de su nombre.

* A partir de **PHP 5.0** se introduce en el lenguaje una característica que permite especificar el tipo de dato esperado para los parámetros de una función o método llamada **type-hinting.**
* Del mismo modo, en **PHP 7.0** se permite especificar el tipo de dato que se debe devolver. Esto ayuda a mejorar la claridad y la seguridad del código al indicar de manera explícita qué tipos de datos se pueden usar en una función o método específico.

Hay dos aspectos principales del type hinting en PHP:

* Type Hinting para **Parámetros**:

Al definir una función o método, puedes indicar el tipo de dato esperado para los parámetros. Si se pasa un tipo de dato incorrecto, PHP genera un error en tiempo de ejecución si se configura con el flag de "tipos estrictos".

|  |
| --- |
| declare(strict\_types = 1);  function suma(int $a, int $b) {  return $a + $b;  } |

* Type Hinting para el **Valor de Retorno**:

Esto ayuda a garantizar que la función siempre produzca un resultado del tipo especificado. Si la función no devuelve el tipo de dato correcto, PHP generará un error en tiempo de ejecución.

|  |
| --- |
| declare(strict\_types = 1);  function obtenerEntero(): int {  return 42;  } |

## 3.3.- Ámbito de utilización de las variables.

En PHP puedes utilizar variables en cualquier lugar de un programa. Si esa variable aún no existe, la primera vez que se utiliza se reserva espacio para ella. En ese momento, dependiendo del lugar del código en que aparezca, se decide desde qué partes del programa se podrá utilizar esa variable. A esto se le llama **visibilidad de la variable.**

Si la variable aparece por primera vez dentro de una función, se dice que esa **variable es local** a la función. Si aparece una asignación fuera de la función, se le considerará una **variable distinta**.

Si en la función anterior quisieras utilizar la variable $a externa, podrías hacerlo utilizando la palabra **global**.

De esta forma le dices a PHP que no cree una nueva variable local, sino que utilice la ya existente.

|  |
| --- |
| $a = 1;  function prueba(){  global $a;  $b = $a; // En este caso se le asigna a $b el valor 1  return $b;  }  var\_dump(prueba()); |

Para trabajar con variables globales también podemos usar el **array** asociativo: **$GLOBALS**.

|  |
| --- |
| $a = 23;  function prueba(){  $a=50;  echo $a . "<br>"; //mostrará el valor local de a, es decir 50  echo $GLOBALS["a"]; //mostrará el valor global de a, es decir 23  }  prueba(); |

Las variables locales a una función desparecen cuando acaba la función y su valor se pierde. Si quisieras mantener el valor de una variable local entre distintas llamadas a la función, deberás declarar la variable como estática utilizando la palabra **static**.

|  |
| --- |
| function contador(){  static $a=0;  $a++; // Cada vez que se ejecuta la función, se incrementa el valor de $a  return $a;  }  var\_dump(contador());  var\_dump(contador());  var\_dump(contador());  var\_dump(contador()); |

Las variables estáticas deben inicializarse en la misma sentencia en que se declaran estáticas. De esta forma, se inicializan sólo la primera vez que se llama a la función.

## 3.4.- Funciones anónimas y funciones flecha.

Existe la posibilidad de definir funciones sin asociarles un nombre con el que invocarlas posteriormente.

|  |
| --- |
| function () {  return "Hola mundo";  }; |

Podemos almacenar la función, como si fuera un dato más, en una variable y utilizar el nombre de la variable para invocar a la función.

|  |
| --- |
| $holaMundo = function () {  return "Hola mundo";};  echo $holaMundo();// Fíjate en los paréntesis del final que indican que estoy invocando la función |

Podemos seguir usando la función como un dato que se pasa como parámetro a otra función. Las funciones que se reciben funciones como parámetros para utilizarlas dentro de su lógica se llaman **funciones de alto orden**.

|  |
| --- |
| function saludar ($saludo) {  echo $saludo();  }  saludar(function () { // Invocación a la función saludar pasando la función anónima como parámetro  return "Hola mundo";  }); |

Es posible acceder a variables definidas en un contexto superior desde una función anónima importándolas con la palabra reservada use y entre paréntesis el nombre de las variables.

|  |
| --- |
| $cadena = "Hola mundo";  $saludo = function() use ($cadena) {  echo $cadena;  };  $saludo(); |

Además, las funciones anónimas aceptan parámetros de entrada como ocurre con las funciones normales.

|  |
| --- |
| $mensaje = "Hola mundo";  $saludo = function($mens) {  echo $mens;  };  $saludo($mensaje); |

Existe un tipo de sintaxis alternativa para definir funciones anónimas disponible a partir de la **versión 7.4** de PHP. Se trata de un modo más conciso y simplificado de crear funciones. Las funciones flecha tienen la forma básica: **fn (argument\_list) => expr**.

|  |
| --- |
| $cubo = fn($x) => pow($x,3);  echo $cubo(4); |

Las funciones flecha soportan las mismas características que las funciones anónimas, excepto que el uso de variables del ámbito padre siempre es automático sin necesidad de importarlas.

|  |
| --- |
| $y = 1;  $suma1 = fn($x) => $x + $y; // Usando función de flecha con acceso al contexto padre  $suma2 = function ($x) use ($y) { // Usando función anónima importando la variable  return $x + $y;  };  echo $suma1(3), "</br>"; // Usamos la función flecha  echo $suma2(6); // Usamos la función anónima |

## 3.4.- Funciones de array de alto orden.

El repertorio de funciones de array incluye un grupo de funciones que admite funciones como parámetros. Este tipo de funciones se denomina funciones de alto orden.

Las funciones son las siguientes:

* **array\_map**(callable $callback, array $array1, array $... = ?): array Aplica la función recibida como parámetro a los elementos de los arrays dados.
* **array\_filter**(array $array, callable $callback = ?, int $flag = 0): array Filtra elementos de un array usando una función recibida como parámetro.
* **array\_reduce**(array $array, callable $callback, mixed $initial = null): mixed Reduce iterativamente un array a un solo valor usando una función recibida como parámetro.
* **usort**(array &$array, callable $value\_compare\_func): bool Ordena una array según sus valores usando la función recibida como parámetro

## 3.5.- Inclusión de ficheros externos.

Conforme vayan creciendo los programas que hagas, resulta trabajoso encontrar la información que buscas dentro del código. Resulta útil agrupar ciertos grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, puedes hacer referencia a esos ficheros para que PHP incluya su contenido como parte del programa actual.

Para incorporar contenido de un archivo externo, tienes varias posibilidades:

* **include:** Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual, en **el mismo punto** en que se realiza la llamada. La ubicación del fichero puede especificarse utilizando una ruta absoluta, pero lo más usual es con una **ruta relativa**. En este caso, se toma como base la ruta que se especifica en la directiva **include\_path** del fichero **php.ini**. Si no se encuentra en esa ubicación, se buscará también en el directorio del guion actual, y en el directorio de ejecución.
* **include\_once:** Si por equivocación incluyes más de una vez un mismo fichero, lo normal es que obtengas algún tipo de error (por ejemplo, al repetir una definición de una función). **include\_once** funciona exactamente igual que **include**, pero **solo** incluye aquellos ficheros que aún no se hayan incluido, hay que tener cuidado de no abusar de esta opción pues es mucho más pesada al tener que comprobar una y otra vez los ficheros incluidos.
* **require:** Si el fichero que queremos incluir no se encuentra, **include** da un aviso y continua la ejecución del guion. La diferencia más importante al usar **require** es que en ese caso, cuando no se puede incluir el fichero, se **detiene la ejecución** del guion.
* **require\_once**. Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo, al igual que **include\_once** y por los mismos motivos no conviene abusar de esta opción.

Cuando se comienza a evaluar el contenido del fichero externo, se abandona de forma automática el modo PHP y su contenido se trata en principio como etiquetas HTML. Por este motivo, es necesario **delimitar** el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores **<?php y ?>.**

Muchos programadores utilizan la doble extensión **.inc.php** para aquellos ficheros en lenguaje PHP cuyo destino es ser incluidos dentro de otros, y nunca han de ejecutarse por sí mismos.

# 4.- Manejo de ficheros

En PHP, el manejo de archivos es esencial para realizar tareas como lectura, escritura y manipulación de datos almacenados en archivos.

**Apertura y Cierre de Archivos**

La función principal para trabajar con archivos en PHP es **fopen(),** que se utiliza para abrir un archivo. Después de realizar operaciones en el archivo, es importante cerrarlo **con fclose().** Los flags de uso del fichero que se establecen en la apertura del fichero cambian dependiendo de si el fichero contiene caracteres o bytes, es decir, si el fichero es de **texto o binario.**

**Lectura de Archivos**

Para leer el contenido de un archivo, puedes usar funciones como **fread()** para leer una cantidad concreta de bytes y **fgets()** para leer los caracteres hasta el final de la línea o hasta que se alcanza el límite especificado.

**Escritura en Archivos:**

Para escribir en un archivo, puedes usar funciones como **fwrite()** o **fputs()** para escribir una cadena o un conjunto de bytes en un archivo y **file\_put\_contents()** para abrir, escribir un bloque de bytes, y cerrar el fichero en una única operación.

**Lectura de Todo el Contenido a la Vez:**

Si solo necesitas leer el contenido completo de un archivo, puedes usar **file()** para leer los contenidos de un fichero en un array y **file\_get\_contents()** para transmitir todo el contenido de un fichero a una cadena.

Estos son solo algunos ejemplos básicos. PHP proporciona muchas mas funciones para el manejo de archivos, como **file\_exists()** para comprobar la existencia de un fichero, **feof()** para detectar el final del fichero, **unlink()** o **delete()** para eliminar archivos, entre otras.

Mas información: <https://www.php.net/manual/es/book.filesystem.php>

# 5.- Formularios web.

La forma natural para hacer llegar a la aplicación web los datos del usuario desde un navegador, es utilizar **formularios**.

Se utilizan las etiquetas **<form> </form>,** dentro de un formulario se incluyen los elementos sobre los que puede actuar el usuario, usando las etiquetas <input> <select> <textarea> y <button>.

El atributo **action** del elemento **form** indica la página a la que se le enviarán los datos del formulario. Script PHP.

El atributo **method** especifica el método usado para enviar la información. Dos valores:

* **GET**: con este método los datos del formulario se agregan al URI utilizando un signo de interrogación “?” como separador, si hay varios se separan por “&”.
* **POST**: con este método los datos se incluyen en el cuerpo del formulario y se envían utilizando el **protocolo** **HTML**.

El proceso de envío y recepción de los datos desde el cliente al servidor es muy importante el atributo **name** del elemento HTML que va a recoger la información del usuario. El valor de dicho atributo servirá de índice para acceder al valor cuando se ejecute el script PHP en el servidor.

# 5.1.- Procesamiento de la información devuelta por un formulario web.